DOCUMENT RESUME

ED 469 185 TM 034 490

AUTHOR	Green, Bert F.
TITLE	Fixed-Weight Methods of Scoring Computer-Based Adaptive Tests. Computerized Testing Report. LSAC Research Report Series.
INSTITUTION	Law School Admission Council, Newtown, PA.
REPORT NO	LSAC-R-97-12
PUB DATE	2002-07-00
NOTE	43p.
PUB TYPE	Numerical/Quantitative Data (110) -- Reports - Research (143)
EDRS PRICE	EDRS Price MF01/PC02 Plus Postage.
DESCRIPTORS	*Adaptive Testing; Bayesian Statistics; Computer Assisted Testing; Computer Simulation; Maximum Likelihood Statistics; *Scoring; Test Items
IDENTIFIERS	*Weighting (Statistical)

ABSTRACT

Maximum likelihood and Bayesian estimates of proficiency, typically used in adaptive testing, use item weights that depend on test taker proficiency to estimate test taker proficiency. In this study, several methods were explored through computer simulation using fixed item weights, which depend mainly on the item's difficulty. The simpler scores involve averages of the weights of the items answered correctly, sometimes also including average weights for the items answered incorrectly. Results show that the root mean squared error for the fixed-weight scores are somewhat larger than those for maximum likelihood scores. The fixed item weights require some adjustment to estimate extreme values of proficiency accurately. Three appendixes contain details of scoring procedures, quadratic adjustments to extreme item weights, and a set of computer programs for simulations. (Contains 10 figures, 3 tables, and 4 references.) (Author/SLD)

Reproductions supplied by EDRS are the best that can be made
from the original document.

ED 469 185

LSAC RESEARCH REPORT SERIES

TM

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL HAS BEEN GRANTED BY

J. VASELECK

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

1

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☑ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

■ Fixed-Weight Methods of Scoring Computer-Based Adaptive Tests

Bert F. Green
Johns Hopkins University

■ Law School Admission Council
Computerized Testing Report 97-12
July 2002

TM034490

A Publication of the Law School Admission Council

BEST COPY AVAILABLE

The Law School Admission Council is a nonprofit corporation that provides services to the legal education community. Its members are 200 law schools in the United States and Canada.

Copyright © 2002 by Law School Admission Council, Inc.

All rights reserved. No part of this report may be reproduced or transmitted, in any part or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission of the publisher. For information, write: Communications, Law School Admission Council, Box 40, 661 Penn Street, Newtown, PA 18940-0040.

LSAT® and LSAC are registered marks of the Law School Admission Council, Inc.

This study is published and distributed by the Law School Admission Council (LSAC). The opinions and conclusions contained in these reports are those of the authors and do not necessarily reflect the position or policy of the Law School Admission Council.

Table of Contents

Executive Summary 1

Abstract 1

Introduction 1

CAT Scoring With Fixed-Weights 3

Simulations 5

Procedure 5

Results 6

Comments 10

References 11

Appendix A: Details of Scoring Procedures 12

Appendix B: Quadratic Adjustments to Extreme Item Weights 13

Appendix C: Complete Set of Programs for Simulations 15

C1: Main Simulation Program 16

C2: IRT Subroutines 24

C3: Scoring Programs 28

C4: General Programs and Test Definitions 35

Executive Summary

Computer-based adaptive tests (CATs) are beginning to replace traditional paper-and-pencil tests. Several major standardized tests are now given as CATs, and the Law School Admission Council (LSAC) is investigating the feasibility and potential benefits of computerizing the Law School Admission Test (LSAT). Not only are CATs more efficient than traditional tests, but they are more popular among test takers. CAT gains its efficiency by tailoring the difficulty of the items given to a test taker, based on that test taker's response to earlier items on the test. Scoring a CAT is complex; the number of items answered correctly is not an appropriate test score because the difficulty of the items must be taken into account. Indeed, most test takers answer about the same number of items correctly. Currently available CAT scoring algorithms use either maximum-likelihood or Bayesian procedures, which are statistically elegant, but extremely difficult to explain to a nonstatistician. The present research project examines a simpler way of scoring a CAT.

Current methods of scoring CATs, including maximum-likelihood estimation and equated number-right, estimate the score (or *proficiency*) of the test taker as a weighted combination of item scores. The weights for the items for these methods depend not only on item characteristics but on the proficiency of the test taker. A correctly answered item can weigh heavily in one test taker's score, and the same item, also correctly answered, can have very little weight in another test taker's score, depending on the other item responses given by those test takers. Such differential treatment is statistically optimal, but it is not easily explained to test takers. The methods proposed here give a fixed-weight to each item, based only on item difficulty, and will be called fixed-weight scores.

Computer simulations of CATs were done in order to compare the fixed-weight scores with the more sophisticated scores (e.g., maximum-likelihood and equated number-right). Two different item pools were used: a "flat" pool with item difficulties uniformly distributed across the scale, and a "special" pool with item parameters more closely matching those found in LSAT item pools. For each of 26 different levels of proficiency, 2,500 simulated test takers responded to a 30-item CAT. Each CAT was scored by each method, and the results were compared.

All scoring methods, including the new fixed-weight scoring methods, provided statistically unbiased estimates of test taker proficiency. The precision, as assessed by the root mean squared error, was somewhat poorer for the fixed-weight scores than for the statistically efficient scores provided by maximum-likelihood and equated number-right scoring. Errors were about 20% larger for the fixed-weight scores. A fixed (nonadaptive) test of the same length, scored by maximum-likelihood, has a measurement error of about 60% higher than a CAT score by maximum-likelihood. That is, most of the advantage of adaptive testing is retained if the simpler fixed-weight scoring system is used. Since the fixed-weight scores are very highly correlated with the maximum-likelihood and equated number-right scores, one can consider maximum-likelihood and equated number-right scores to be statistically refined versions of the proposed fixed-weight methods.

Abstract

Maximum-likelihood and Bayesian estimates of proficiency, typically used in adaptive testing, are difficult to explain to nonexperts. These methods use item weights that depend on test taker proficiency to estimate test taker proficiency. Several methods have been explored using fixed item weights, which depend mainly on the item's difficulty. The simpler scores involve averages of the weights of the items answered correctly, sometimes also including average weights for the items answered incorrectly. Results show that the root mean squared errors for the fixed-weight scores are somewhat larger than those for maximum-likelihood scores. The fixed item weights require some adjustment to accurately estimate extreme values of proficiency.

Introduction

The Law School Admission Council (LSAC) is considering the possibility of administering the Law School Admission Test (LSAT) on the computer and possibly in an adaptive format. CATs are now being used in many testing programs, including the Armed Services Vocational Aptitude Battery (ASVAB), the Graduate Record Examination (GRE), and others (Wainer, 1990). CATs are well received by most test takers, who like both the computer administration and the adaptive process, which is relatively easy to understand. Many test takers actually report that they enjoy taking such a test.

The scoring procedure for CAT is much different from the scoring procedure of a conventional test in which all test takers respond to the same items. The conventional score, the number of items answered correctly, is not a reasonable index of a test taker's ability on a CAT. An adaptive test chooses items one at a time for each test taker; the difficulty of each item depends on the test taker's success or lack of success with

earlier items. As a consequence, most test takers tend to answer about the same number of items correctly, although item difficulty differs. It would seem obvious that the test score must take into account the difficulty of the particular test items for each test taker.

The procedures used in CAT are based on item response theory (IRT). In IRT, the test score is viewed as an estimate of an underlying ability or proficiency, θ, of the test taker. IRT represents the probability that a test taker of proficiency θ would correctly answer item i as

$$P_i(\theta) = c_i + \frac{1-c}{1+e^{-1.7a_i(\theta-b_i)}}, \tag{1}$$

where a_i, b_i, and c_i are discrimination, difficulty, and lower asymptote parameters associated with item i, respectively. The function is called the item characteristic curve (ICC) for the item. Equation 1 provides the three-parameter logistic (3-PL) version of the ICC, details of which may be found in Lord (1980). The probability of an incorrect response is given as

$$Q_i(\theta) = 1 - P_i(\theta). \tag{2}$$

Three procedures for estimating θ, and thereby scoring a CAT, are in common use. One is the widely used maximum-likelihood estimate of proficiency. The other two involve Bayesian methods (using either the mode or the mean of the posterior proficiency distribution as described below). All three methods base the score on the response to each item, together with the item's characteristic curve. Both correct and incorrect responses influence the score—what a test taker cannot answer correctly is as informative as what the test taker can answer correctly.

The maximum-likelihood method involves the likelihood, which is a function of θ. For a given value of θ, the likelihood represents the joint probability of the particular pattern of correct and incorrect responses to the particular items in the test taker's test. The set of responses given by a certain test taker to a set of n items can be represented by the variables u_i, for $i = 1$ to n, where $u_i = 1$ if the response was correct, and $u_i = 0$ if the response was incorrect. The likelihood, L, is given by

$$L = \prod_{i=1}^{n} P_i^{u_i} \left(1 - P_i\right)^{1-u_i}. \tag{3}$$

The maximum (mode) of this likelihood function is the value of θ for which the pattern of responses is most likely. Thus the maximum-likelihood estimate is found by solving

$$\sum_{i=1}^{n} \left[u_i - P_i(\theta)\right] \frac{P_i'(\theta)^2}{P_i(\theta)Q_i(\theta)} = 0, \tag{4}$$

where

$$P_i'(\theta) = \frac{\partial P_i}{\partial \theta}.$$

Bayesian estimation combines the likelihood (Equation 3) with an assumed prior distribution of proficiency (usually the standard normal distribution) and yields a posterior distribution of the test taker's true θ. The posterior distribution represents the uncertainty of the estimate of the test taker's proficiency. The mode or the mean of the posterior distribution provides the estimate of θ, while its standard deviation provides the standard error of measurement. The mode of the posterior distribution is called the Bayes modal estimate; the mean is called the expected *a posteriori* estimate.

Recently, Stocking (1996) proposed an alternative method for scoring CATs, which she calls the equated number-right score. Her proposal is to use the responses on the CAT to estimate the number of items the test taker would have answered correctly if he or she had responded to the entire item pool (or a reference set of items). The equated number-right score bears almost no relationship to the number of items answered correctly on the CAT. Nearly all test takers have about the same number of correct responses on a CAT. In equated number-right scoring, the test taker's item responses, both rights and wrongs, together with the item characteristics, are used to obtain a score on the θ metric. Then θ is translated onto the observed-score metric (expected number-correct score) for the full reference item pool. The equation that is first solved for θ is

$$\sum_{i=1}^{n} [u_i - P_i(\theta)] = 0. \quad (6)$$

This step turns out to be a minor variant of maximum-likelihood scoring, the difference being that the items are not weighted by their information functions, as happens in maximum-likelihood scoring. The subsequent step of converting θ to an estimated number-correct score is explored in Appendix A.

It is not difficult to see why psychometricians shrink from explaining the scoring algorithms to lay persons. Test takers would have to be told that their scores represent ability level that was the most likely score (maximum-likelihood), the most probable score (Bayes modal), or the expected score on the entire item pool (equated number-right). Further, it would have to be explained that their scores were based on the pattern of right and wrong answers that they gave to the items on their particular test, and on the difficulty of those items.

The most troublesome aspect of any of these methods of scoring CATs is that the amount of credit earned for either a correct or an incorrect answer depends both on the item difficulty and on the test taker's estimated proficiency (θ). This is not only difficult for test takers to grasp, but it is also difficult for the program to implement. Having item weights that depend on proficiency, when proficiency is being estimated, necessarily results in an iterative computation. In order to estimate proficiency, one must know the item weights, but in order to know the weight for a given item response, one must know the test taker's proficiency. The scoring algorithm uses an initial estimate of proficiency to compute the weights, which leads to a better estimate of the score. The weights are adjusted to the new estimate, and the iterative process continues until it converges. In fact, from time to time, the maximum-likelihood and equated number-right scores may not converge. For example, when all the responses are correct, the estimate diverges to an indefinitely large value. When all the responses are wrong, the estimate diverges to an indefinitely small score (i.e., a large negative value). In practice, of course, minimum and maximum scores are established. The Bayesian methods are not prone to this indeterminacy, which is why they are often preferred. The iterative calculations can be done in the blink of an eye these days, but explaining the method takes longer.

Test takers can readily understand that more credit is earned for correctly answering more difficult questions in an adaptive test. Less obvious is the fact that some differential credit is also given for failing to answer correctly. That is, failing an item is informative. This is also true of classical test theory scoring, where the test raw score is generally the number of items answered correctly. Number right really means proportion correct out of a fixed number of items, since items not answered are treated as incorrect. Omitting is not permitted in CAT, and items-not-reached are discouraged, whenever possible, by liberal time limits. CAT scores are based on the items actually answered; special treatment of items not answered is optional. In classical test scoring, all correct answers to an item earn the same credit, regardless of their difficulty. Of course, in classical test scoring, it is assumed that all test takers take the same items, which is not true in adaptive testing.

In maximum-likelihood, Bayes modal, expected *a posteriori*, and equated number-right scoring, the same item answered correctly still counts differently for different test takers. The effect of each item response depends on the subset of items answered by the test taker, as well as which of these items were answered correctly. Variable item weights is not a property of the procedures that are proposed here. In the proposed methods, every item contributes the same amount to the score of everyone who answers it correctly. The scoring schemes that are to be suggested here use different weights for different items, but they do not use item weights that are dependent on the test taker's proficiency level. Instead, the weights depend solely on item characteristics. The fixed-weight methods do retain the idea that both correct and incorrect item responses influence test score.

CAT Scoring With Fixed-Weights

Early in the development of CAT, successive items were selected by stepping the difficulty up after a correct response, and stepping the difficulty down after an incorrect response. Large steps were used at first; smaller steps in difficulty were used as the test progressed. The final score was the difficulty of the item answered last (Lord, 1970), but the average difficulty of the items administered was also considered. This procedure is very similar to the up-and-down methods used in psychophysics to determine absolute and differential limens for sound, light, and other sensory stimuli. The up-and-down methods of sensory sensitivity have been well worked out and have proven very satisfactory. Similar methods are investigated here for CAT scoring.

The simplified scoring schemes explored in this paper can be viewed as approximations of the statistically elegant methods, but they are expected to be easier for test takers to understand and for test users to explain and justify. The schemes involve giving different weights (different amounts of credit) to the items, based essentially on their difficulty, just like the old quiz show that had $16 questions and $64 questions. In the simplest scheme, the test taker's score is the average of the weights for the items she or he answered correctly. In other schemes, partial credit is given for wrong responses.

The main question is what weights to use. Several possibilities were considered. The item difficulty parameter, b, might be used. The θ for which the item response function was 0.5 could also be used. The weight finally chosen was based on the θ for which the item information was maximum (θ_{max}). In the 3-PL IRT model, this is

$$\theta_{max} = b + \frac{1}{1.7a} \ln\left[0.5 + \sqrt{0.25 + 2c}\right]. \tag{7}$$

θ_{max} merely adjusts b depending on c. If $c = 0$, there is no adjustment and the weight for an item is b. θ_{max} seemed a better value on which to base the item weight than just b. It was quickly found that scores higher than 1.5 were not high enough, and scores below −1.5 were not low enough. This is demonstrated in Figure 1 for one of the fixed-weight scoring methods.

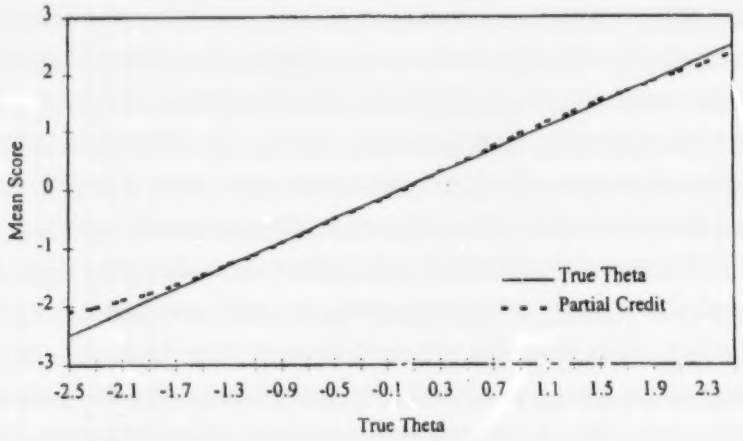

FIGURE 1. *Mean partial-credit (fixed-weight) scores using θ for maximum information (θ_{max}) as weights, plotted with the true θ values (solid line)*

Consequently, in those regions (greater than 1.5 and less than −1.5), a quadratic adjustment was devised for the item weights. The new weights are shown in Figure 2. The weights remain θ_{max} between −1.5 and 1.5, but are given by simple quadratic functions of θ_{max} for the extremes. The upper and lower extremes have slightly different quadratics, because of the differential effect of the c parameters. Details are given in Appendix B.

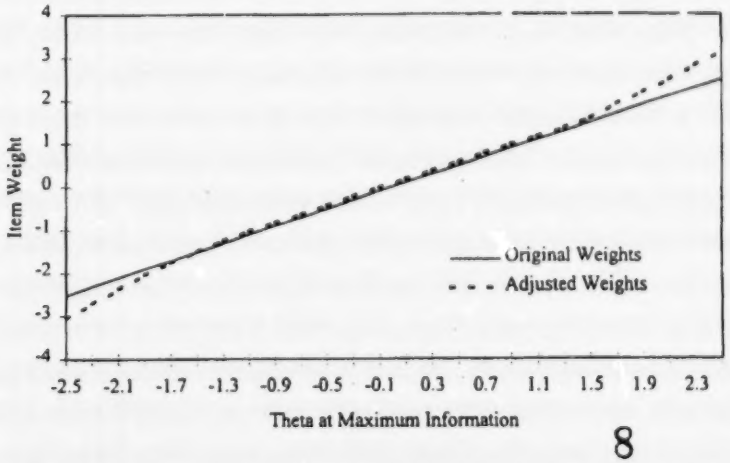

FIGURE 2. *Item weights as a function of θ for maximum information (θ_{max}). The original (nonadjusted) values are also shown*

Using the adjusted item weights shown in Figure 2, the following scores were examined:

- Average Weights (Rights), or ave_right = average item weight for items answered correctly
- Average Weights (Wrongs), or ave_wrong = average weight for items answered incorrectly
- Partial Credit = (ave_right + p*ave_wrong)/(1.0 + p) where p = 0.5 in these studies (i.e., average weights of rights + half weights of wrongs)
- Average Weights (All) = (*n*right * ave_right + *n*wrong * ave_wrong) / (*n*right + *n*wrong), where *n*right and *n*wrong are the number of items answered correctly and incorrectly, respectively (i.e., equal weights for rights and wrongs)

There is no practical interest in using average weights (rights) or average weights (wrongs), but they were examined for possible clues about the performance of partial credit and average weights (all). The latter two actually perform about equally well, in general. The results for average weights (rights) and average weights (wrongs) closely paralleled the results for partial credit and average weights (all) and were otherwise uninformative, so these scores are not considered further here.

In principal, the average weights (all) score, which treats rights and wrongs equally, is in many respects like the equated number-right, maximum-likelihood, and Bayes modal scores. However, average weights (all) scores depend more on the item selection sequence than the other fixed-weight scores listed above. It is possible that using some kind of trimmed averages, with strong trimming, might improve the precision of the scores. This might be acceptable to test takers, but it may seem a bit counter to the spirit of scoring with item weights.

Simulations

Procedure

Several simulations have been performed to examine the utility of the fixed-weight scoring methods. Although all four fixed-weight scores were calculated, two were considered at length: partial credit and average weights (all). The latter is sensible in the psychophysics application (up-and-down method) but may depend overly much on the adaptive algorithm. In addition, maximum-likelihood estimates and equated number-right scores were obtained.

The simulations used two pools, each with 300 simulated items. The "flat item pool" used *b*'s uniformly distributed in the range –2.5 to 2.5, *a*'s in the range 0.75 to 1.25, and all *c*'s at 0.15. The second item pool, called the "special item pool," had a negatively skewed distribution of *a*'s, ranging from 0.4 to 1.7; the *b*'s still ranged from –2.5 to 2.5, and the *c*'s were all 0.15. The special item pool was designed to reflect *a*-values similar to those found in IRT parameterizations of actual LSAT item pools. On the other hand, the range of *b*-values found on the LSAT was consistent with the flat item pool, so the *b*'s were not altered for the special item pool.

All simulations used a 30-item CAT. At the start of each CAT, estimated θ was arbitrarily stepped a large amount until the simulated test taker made at least one correct and one incorrect response. From then forward, estimated θ was based on an approximation to Bayes modal estimation, while item selection was based on maximum information. No procedure for controlling item exposure was used, nor was content balancing simulated. All simulated test takers "answered" all 30 items; simulated responses followed a 3-PL IRT model given in Equation 1. It was assumed that item parameters were the true population values; there was no provision for errors in estimated parameters. All simulations consisted of 2,500 replications of CAT (i.e., 2,500 simulated test takers) at each of 26 values of true θ (from –2.5 to 2.5 in increments of .2). Each CAT was scored using maximum-likelihood, equated number-right, and the four scoring methods listed above based on fixed item weights. In the unlikely case that all 30 responses were of one kind, maximum-likelihood and equated number-right used an estimated θ of either –3.0 or 3.0, as appropriate. For each θ level, the mean, standard deviation, and root mean squared error (RMSE) were calculated, where error is defined as the difference between the score (estimated θ) and the actual (true) θ. Then a linear equating was performed, and the same quantities were recomputed. Results are presented graphically for mean scores and RMSE. For summary purposes, a weighted average of RMSE was calculated, using unit normal weights.

Appendix C contains a copy of the final simulation program, including all subroutines.

Results

The mean scores for maximum-likelihood, equated number-right, partial credit, and average weights (all) are shown as a function of true θ in Figure 3 for the flat item pool, and in Figure 4 for the special item pool. The true θ values are recovered with very little bias, on the average, although there are still vestiges of the curvature (bias) in the extremes.

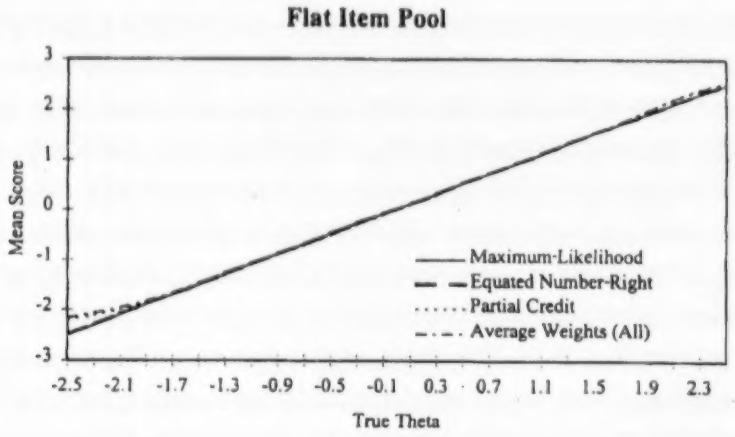

FIGURE 3. *Average scores for 2,500 simulated test takers at each of 26 true θ values, for four scores: maximum-likelihood, equated number-right, partial credit weights, and average weights (all). CATs drawn from a flat item pool.*

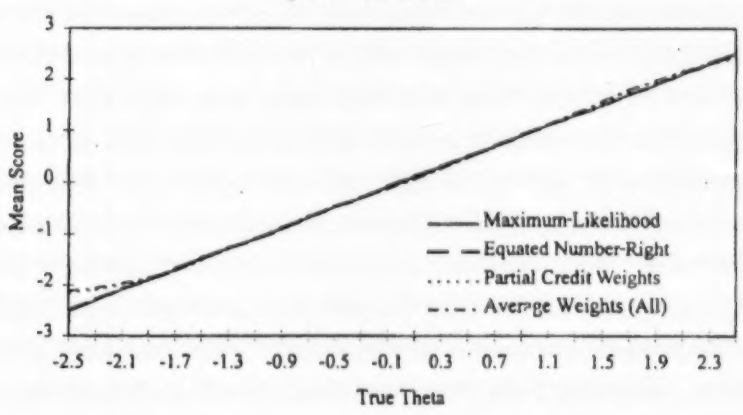

FIGURE 4. *Average scores for 2,500 simulated test takers at each of 26 true θ values, for four scores: maximum-likelihood, equated number-right, partial credit weights, and average weights (all). CATs drawn from a special item pool*

The root mean squared errors (RMSEs) are shown for the flat item pool in Figure 5 and for the special item pool in Figure 6. As shown in Figure 5, the equated number-right score is nearly as good as the maximum-likelihood score, and RMSE for scores with fixed item weights are about 10% to 30% larger than the maximum-likelihood score. For the special item pool (Figure 6), the fixed-weight scores (average weights —all and partial credit) are about as good as the maximum-likelihood and equated number-right scores at several θ values, but have considerably more error for other θ values. Table 1 shows the weighted averages of RMSE for the various scores for both the flat and special item pools.

TABLE 1
Weighted average root mean squared error (RMSE) for several methods of CAT scoring (weights based on normal distribution)

Scoring Method	CATs From Flat Item Pool	CATs From Special Item Pool
Maximum-likelihood	0.250	0.271
Equated number-right	0.253	0.276
Partial credit weights	0.297	0.303
Average weights (all)	0.305	0.310
Average weights (rights)	0.305	0.310
Average weights (wrongs)	0.298	0.309

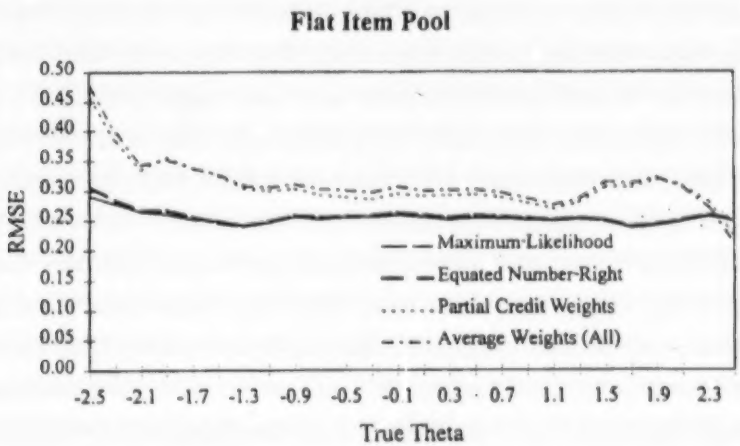

FIGURE 5. *Root mean squared errors (RMSEs) for scores shown in Figure 3*

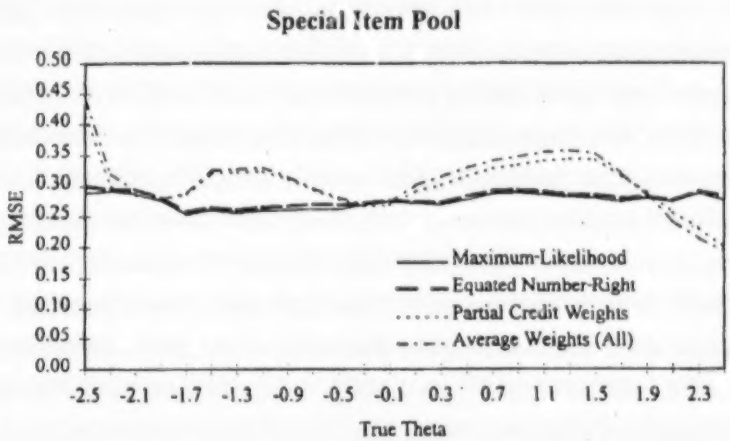

FIGURE 6. *Root mean squared errors (RMSEs) for scores shown in Figure 4*

BEST COPY AVAILABLE

The fixed-weight scores have less variability than maximum-likelihood scores for extremely high and extremely low θ values. This puzzling anomaly apparently results from the treatment of cases with all incorrect or all correct responses. Algorithms for both maximum-likelihood and equated number-right must be set to give some arbitrary score in that event, since the iterative formulas would otherwise diverge without limit. (Although equated number-right scoring estimates the number-correct score on the entire pool, it does so through the mechanism of first estimating θ, and thus has the same problem as maximum-likelihood with the all-correct and all-incorrect response patterns.) The choice here was to use –3.0 or +3.0, as appropriate. When, instead, the limiting values were set to –2.5 and +2.5, the RMSEs of the maximum-likelihood and equated number-right scores were smaller than those for the fixed-weight scores at both extremes, as they should have been. This result shows one way in which Bayesian scores would be superior to maximum-likelihood and equated number-right scores.

Some dead ends and side issues should be reported. Because of the somewhat unusual pattern of RMSEs, some attempts were made to see if the source could be found. First, it was hypothesized that the item pool may not have contained enough items at the extremes. Consequently the flat item pool was extended to *b* values ranging from –3.5 to +3.5. Such pools are virtually impossible in practice, but they are easily simulated. No appreciable change in the pattern of RMSE was seen. Then, it was thought that the trouble might lie with not disregarding the early item responses before the test taker reached the appropriate level of item difficulty. To get around that, simulations were run in which the CAT was always started at the test taker's true θ value. Figure 7 shows the resulting RMSE's values for the flat item pool. It can be seen that they are slightly lower than those in Figure 5, but the pattern remains the same.

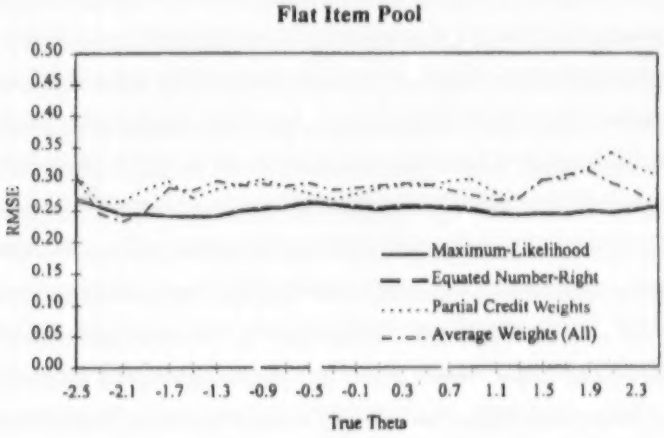

FIGURE 7. *Root mean squared errors (RMSEs) for scores obtained with the flat item pool, using the simulated test taker's true θ (rather than 0) as a starting point for that test taker's CAT*

One unexpected result is that the fixed-weight method turns out to be inappropriate for traditional linear (i.e., nonadaptive) tests, in which all test takers answer the same items. The effects of using the fixed-weight method of scoring was tried on a linear test of 30 items. The main purpose of this analysis was to compare the slight difference in accuracy among scoring methods for the CATs with the large difference to be expected between a CAT and a linear test.

The fixed-weight scoring method is specifically designed for adaptive testing. Like the maximum-likelihood and equated number-right methods, it is designed to use the information about which items were answered. However, the fixed-weight method apparently depends more on this information than the other methods. In fact, the method that gives full weight to the wrong responses (average weights—all) essentially takes all its information from which items were administered. In a nonadaptive test, the average weights (all) scoring method gives everyone exactly the same score: the sum of the weights of all items on the test. The only fixed-weight score that might make sense in a linear test is the score based on the items answered correctly (average weight [rights]).

Table 2 provides the weighted RMSE for the maximum-likelihood, equated number-right, average weights (rights), and percent-correct scores for a linear (nonadaptive) test. The fixed-weight scores yield very high measurement error for the very high and very low scores. To examine the problem further, the means,

standard deviations, and RMSE's for these same scores were plotted as a function of θ, as shown in Figures 8, 9, and 10, respectively. (Linear equating to the true θ scale was performed for all of the scores, which is why the percent-correct score can be shown on the θ scale rather than the 0-100% scale.) The fixed-weight average weights (rights) score is very poor in the extremes.

TABLE 2
Weighted average root mean squared error (RMSE) for scores on a linear (nonadaptive) 30-item test

Scoring Method	Fixed Test From Flat Item Pool
Maximum-likelihood	0.401
Equated number-right	0.444
Average weights (rights)	1.175
Percent-correct	0.439

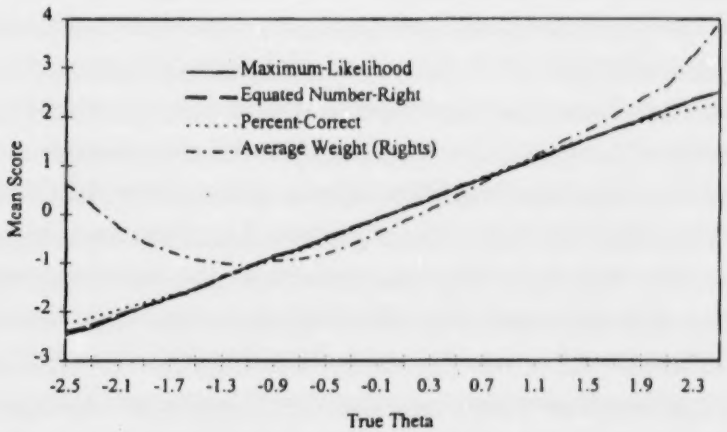

FIGURE 8. *Mean scores for a 30-item linear (i.e., nonadaptive) test, scored via maximum-likelihood, equated number-right, percent-correct, and average weights for all items answered correctly (not all items attempted)*

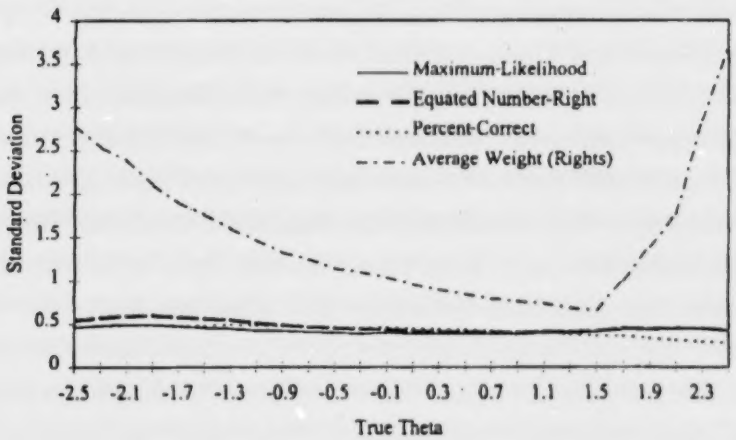

FIGURE 9. *Standard deviation of scores for a 30-item linear (i.e., nonadaptive) test, scored via maximum-likelihood, equated number-right, percent-correct, and average weights (rights) for all items answered correctly (not all items attempted)*

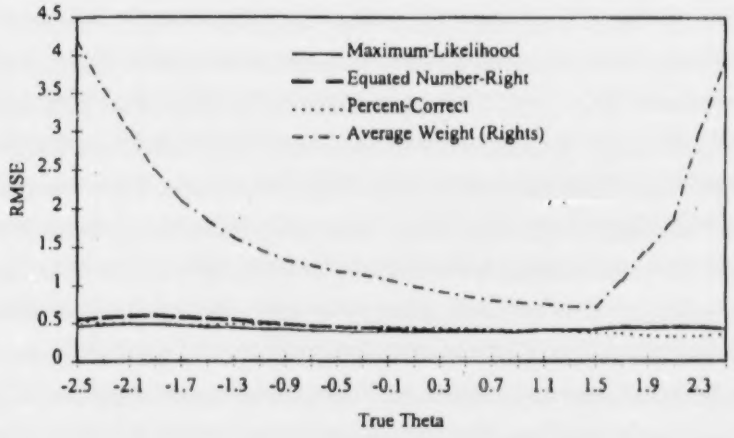

FIGURE 10. *Root mean squared errors (RMSEs) for a 30-item linear (i.e., nonadaptive) test, scored via maximum-likelihood, equated number-right, percent-correct, and average weights (rights) for all items answered correctly (not all items attempted)*

According to the IRT model, a person with a very low proficiency, say $\theta = -2.5$, still has a non-zero probability of correctly answering difficult questions. In a CAT, such a person would not get the chance to try such items, but that chance is available in a linear test. Even if there are, say, 20 items for which the person has a probability of 0.05 or less of answering correctly, the expected number of correct answers is about 1.0. Close inspection of the answer patterns of a number of simulated low-ability test takers shows that nearly all have a correct response for one or more difficult items. Such items would get very little weight from maximum-likelihood or equated number-right scoring methods. However, the fixed-weight method gives an item the same weight for everyone, which elevates the scores of low-ability test takers in this situation. The reverse reasoning holds for the very high-ability test takers, who nevertheless have less than perfect probability of correctly answering easy items. As can be seen in Figures 8, 9, and 10, essentially ignoring responses to items of inappropriate difficulty when calculating average weights (rights) scores might lead to more accurate estimates of proficiency. The percent-correct score uses responses to items of inappropriate difficulty in a non-specific way and is not adversely affected by them. The fixed-weight scoring method, however, is thrown awry by these "anomalous" responses.

Comments

It should be remembered that the correlations among all the scoring methods considered here are well above 0.95 for the particular populations (2,500 cases at each of 26 evenly-spaced θ levels) simulated here. Using a population with normally distributed proficiencies, the correlations are above 0.97. Thus it can be said that the maximum-likelihood score, the equated number-right score, and the Bayesian scores are all refinements on the simple-to-understand fixed-weight scoring methods. That is, a CAT score is essentially a system for giving more credit for more difficult questions answered correctly, coupled with some credit for the difficulty of items answered incorrectly.

The results are based on the usual assumption that the test takers are not trying to "game" the system. It is not clear how the scoring methods would react if a test taker uses a strategy that involves deliberately answering some items incorrectly, in order to obtain an easier test (the Wainer strategy). Another strategy the test taker might adopt is to stop answering items when an item is presented that seems very difficult. The test taker might reason that the computer probably has an overly rosy view of the test taker's proficiency, and the test taker does not wish to threaten this optimistic estimate by making any further wrong responses. The Wainer strategy might fare worse under the fixed-weight method, but the "quit when you think you're ahead" strategy might be a good one. These possibilities have not yet been examined.

All of these results have been evaluated in the metric of IRT, the θ metric. Had they been evaluated in terms of a score equated to a fixed test, or to a reference item pool, the problems at the extremes would have been less evident. A scale metric based on the number-right on a fixed test compresses the extreme IRT proficiencies, or the IRT scale expands the ends of the number-right scale, depending on one's perspective.

It has been suggested that the fixed-weight scores might prove to be more robust in a situation where the one-dimensional IRT model is inappropriate for an item pool. This possibility has not yet been examined.

References

Lord, F. M. (1970). Some test theory for tailored tests. In Holtzman, W. H. (Ed) *Computer-assisted Instruction, Testing, and Guidance*. New York: Harper & Row.

Lord, F. M. (1980). *Applications of item response theory to practical testing problems*. Hillsdale, NJ: Erlbaum.

Stocking, M. L. (1996). An alternative method for scoring adaptive tests. *Journal of Educational and Behavioral Statistics*, 21, 365–389.

Wainer, H. (1990). *Computerized adaptive testing: A primer*. Hillsdale, NJ: Erlbaum.

Appendix A: Details of Scoring Procedures

Maximum Likelihood

The maximum likelihood estimate is found by solving:

$$\sum_{i=1}^{n} [u_i - P_i(\theta)] \frac{P_i'(\theta)^2}{P_i(\theta)Q_i(\theta)} = 0.$$

An additional term is added for Bayes modal scoring, to represent the a priori distribution of possible test scores, generally taken to be the standard normal distribution at first. A routine copied (with permission) from D.R.Divgi, provides an efficient computation, using Newton's method. When the maximum is outside the boundary, currently set at (-3.0, +3.0), the value of -3.0 or +3.0 is generated, as appropriate.

Equated Number-Right Score

The estimate for the Stocking "equated number-right" score is found by solving:

$$\sum_{i=1}^{n} [u_i - P_i(\theta)] = 0.$$

If the iterative solution reaches the boundary values of -3.0 or +3.0, the boundary value becomes the score estimate.

Newton's Method

Both scores are readily found using Newton's method. The general form of Newton's method can be expressed as follows. To find the value of x for which f(x) = 0, it is necessary to be able to find f(x) and f'(x) (first derivative of f(x)) for any value of x. Then, beginning with an estimate x_0, a better estimate can be found from

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

For example, to find the square root of **a**, set up $f(x) = x^2 - a$. Some simple math shows that

$$x_1 = 0.5(x_0 + \frac{a}{x_0}).$$

This converges very quickly. For example, if a = 4, and $x_0 = 1$, the successive estimates of **a** are 2.5, 2.05, 2.00061, 2.00000009, etc.

The routines used for equated number-right and maximum likelihood are shown in Appendix C3. The iteration is set to stop when successive estimates differ by less than 0.001.

Appendix B: Quadratic Adjustments to Extreme Item Weights

Originally the weight for an item was defined as the θ value at which the item information function was maximum: θ_{max}. A variety of early attempts at adjusting this weight function using polynomials indicated that the best adjusted weight function had a central portion that was linear (indeed identical) with θ_{max}, but which turned up for positive extreme values, and turned down for negative extreme values. That is, for items with θ_{max}'s above about 2.0, the item weights were larger than θ_{max} whereas items with θ_{max}'s below about 1.5 had item weights substantially less than θ_{max}. Consequently, it seemed better to use a piecewise function for the item weights. For the range $-1.0 < \theta_{max} < 1.0$, the item weight would be θ_{max}, whereas below and above that range, quadratic or cubic polynomials would be used.

For simplicity, quadratic functions were used. For convenience, the item weights are symbolized by y, the θ_{max} values by x, and a point on the function is designated as (x, y). The general derivation below applies to any choice of beginning and end points of the segments. The point at which the segment joins the main function y = x is called (x_0, y_0) and the extreme point is called (x_1, y_1). The polynomial is

$$y = x + \frac{(y_1 - x_1)(x - x_0)^2}{(x_1 - x_0)^2}.$$

The function for the high end was required to pass through the points (1.0, 1.0), where its derivative was 1.0, and (2.5, 3). At the low end, the points were (-1.0, -1.0), where the slope was 1.0, and (-2.5, -3.2). Using the derivation below, the functions for the low and high extremes turned out to be as follows:

low: $-0.311\theta_{max}^2 + 0.378\,\theta_{max} - 0.311$, and
high: $0.222\,\theta_{max}^2 + 0.556\,\theta_{max} + 0.222$.

General Solution

The general mathematical problem was to find a quadratic that included the points (x_0, y_0) and (x_1, y_1) and that had a first derivative (i.e., a slope) of 1.0 at the point (x_0, y_0). y and y' are given by

$$y = ax^2 + bx + c$$
$$y' = 2ax + b.$$

The equations to be satisfied simultaneously are

$$1 = 2ax_0 + b,$$
$$y_0 = x_0,$$
$$y_0 = ax_0^2 + bx_0 + c, \text{ and}$$
$$y_1 = ax_1^2 + bx_1 + c.$$

The first equation yields $b = 1 - 2a\,x_0$. Putting that result for b into the third equation, together with the second equation, yields

$$x_0 = ax_0^2 + (1 - 2ax_0)\,x_0 + c\,,\ \text{which reduces to } c = ax_0^2.$$

Putting these results for b and c into the third equation yields

$$y_1 = ax_1^2 + (1 - 2ax_0\ )x_1 + ax_0^2$$
$$y_1 - x_1 = a(x_1^2 - 2x_0x_1 + x_0^2\ )$$

The final solution is:

$$a = (y_1 - x_1) / (x_1 - x_0)^2$$
$$b = 1 - 2ax_0$$
$$c = a\,x_0^2$$

The following constraint holds:

$$4ac = (1 - b)^2.$$

Note that the polynomial can be written as

$$y = x + a(x - x_0)^2 \text{ or } y = x + \frac{(y_1 - x_1)(x - x_0)^2}{(x_1 - x_0)^2}.$$

Also, given a, b, c, the corresponding points can be found by

$$x_0 = \frac{1 - b}{2a};\ y_0 = x_0\,.$$

Then any other point can be found by $y = x + a\,(x - x_0)$.

Appendix C: Complete Set of Programs for Simulations

Except for the scoring algorithms based on fixed item weights and the routine for equated number-right scoring (EQNR), all of these programs have been used previously in work in my laboratory in the psychology department of the Johns Hopkins University. All programs are written in Borland's version of Pascal, called Turbo-Pascal. All programs run under Microsoft Windows, either Windows 3.1 or Windows 95. Anyone is welcome to use them, but no warranty of accuracy is given or implied.

C1:	Main simulation program	simscr24
C2:	IRT subroutines	irtlibw
C3:	Scoring programs	scorew
C4:	General programs and test definitions	testdef

C1: Main simulation program

```
program simscr24 ;  { version of 2/15/97}
(*
   simulates standard cats using info table item selection based on
   IRT max information at estimated theta
   scoring weights = b except for polynomial extremes,based on polynomial
      transform of theta at max. info
   parameters (changed by recompliing) :
      numpertheta,  lotheta(delta)hitheta  testlength
   Does a batch of numperbatch CATS of fixed length testlength
   at each  theta: lothetat (delta) hitheta
   Each CAT starts at starttheta, steps estimated theta by step,
     then by step = fstep*step until string of responses has
     at least one right, at least one wrong, then shifts to
     owensbayes estimates
     CAT includes exposure control
     when CAT finished, mle, bme, number rght, and weighted scores
     are calculated and stats kept until batch finished, then processed
*)
uses
    wincrt,
    windos,
    irtlibw,
    testdefw,
    {testdefw includes maxtestlength,
      numitemsperlevel for infotable }
    scorew ;
type
    slabel = array[1..6] of string[9] ;
    bt     = array[0..2] of real ;
const
 {for simulation}
    programlabel = 'simscr24' ;
    numpertheta  = 25 ; {simulates numpertheta for each theta}
    lotheta      = -2.5; {simulation does numpertheta }
    delta        =  0.20 ;
    numtheta     = 26 ; { hitheta=lotheta+(numtheta-1)delta}
  {for scoring}
    numscore     =    6  ;
    scorelabel : slabel = ('   mle    ','avbright ','avbwrong ',
                           '   pcl    ','avbrw    ','eqnr     ')  ;
    partcredit  = 0.5 ;

{scoring weights are transforms of theta at max info}
    lotmi = -1.0 ;  {weight limits}
    hitmi = 1.75  ;
  {{constant,linear,quad)}
    lopoly : bt = (-0.311,0.378,-0.311)  ;
    hipoly : bt = (0.222, 0.556, 0.222 )  ;
    tmi0  = 0.073 ;
    tmi1  = 1.0 ;
  {for CAT specification}
    testlength    = 30 ; {should be no more than maxtestlength}
    t12           = 15 ; {half test length, for trouble output}
    step          = 1.0  ; {initial step size}
    fstep         = 0.75 ; {step size reduction factor}
    firstthetacode = 2 ; {1=0,2=truetheta}
    startttheta    = 0.0 ;
```

BEST COPY AVAILABLE

```
      startvar      = 1.0 ;
      truethetamean = 0.0 ;
      truethetavar  = 1.0 ;
      mlepsd :real        = 0.0 ;
      bmepsd :real        = 1.0 ;
      pav    :real        = 0.0 ;

(*    testitem = RECORD          in unit testdefw              *)
(*       itemindex : integer ;                                 *)
(*       response  : integer ;                                 *)
(*    end; {of testitem record def.}                           *)
(*    testlist = ARRAY[1..maxtestlength] OF testitem ;         *)

var
   td : testlist ;
   iteminx, itemresp : integer ;
   truetheta, hitheta, firsttheta, esttheta, estvar : real ;
   mletheta, mlevar, bmetheta, bmevar, fcn          : real ;
   ip, it, iit, itheta, jtheta, ktheta, nitem, nitems  : integer ;
   numright, numwrong                        : real ;
   mlescore, bmescore,wtright,wtwrong,w3right,w3wrong : real ;
   mletroub, bmetroub    : integer ;
   temp  : real ;
   score      : array[1..numscore] of real ;
   meanscore  : array[1..numscore] of real ;
   varscore   : array[1..numscore] of real ;
   rmsqbias   : array[1..numscore] of real ;
   slope      : array[1..numscore] of real ;
   addconst   : array[1..numscore] of real ;
   batmean    : array[1..numscore,1..numtheta] of real ;
   batsd      : array[1..numscore,1..numtheta] of real ;
   eqweight   : array[1..numtheta] of real ;
   weight     : array[1..maxpoolsize] of real ;

procedure MakeItemWeights ;
  var
      rtmi, tmi,tmi2 : real ;
  begin
    for it :=1 to nitems do begin
       rtmi := thetaformaxinfo(item[it].a,item[it].b,item[it].c) ;
       tmi := tmi0 + tmi1*rtmi ;
       tmi2 := sqr(tmi) ;
       if tmi < lotmi then
           weight[it] := lopoly[0]+lopoly[1]*tmi+lopoly[2]*tmi2
         else if tmi > hitmi then
           weight[it] := hipoly[0]+hipoly[1]*tmi+hipoly[2]*tmi2
         else weight[it] := tmi ;
       end ; {item loop}
  end ; {MakeItemWeights}

procedure writesomeitems;
   var
      temp : real ;
   begin
      for it := 1 to nitems do
         if (it div 10)*10 = it then begin
            temp := thetaformaxinfo(item[it].a,item[it].b,item[it].c) ;
            writeln(outfile,it:3,item[it].a:6:3,item[it].b:6:3,
              item[it].c:6:3,temp:6:3,weight[it]:8:3) ;
         end ;
   end;
```

BEST COPY AVAILABLE

```
procedure PickItem (var CurrentTheta : real; var NextItem : integer);
  (picks next item, standard CAT (category not taken into account)
   ItemUse[k] = 0 if item k is not used or requested
              = 1 if item k is requested but denied
              = 2 if item k is used                          )
  var
     kDeny, kTemp : integer;
     i, j, k      : integer;
  begin
     kDeny := 0;
     kTemp := 0;
     i := LevelOfTheta(CurrentTheta);
     j := 0;
    REPEAT
      repeat (Choose next item, k, based on info.)
        inc(j);                 (Keep picking items until one is found)
        k := InfoTable[i,j];    (that hasn't been used or denied yet--)
        if ItemUse[k] = 1       (unless the end of the pool is reached)
          then kDeny := k;
      until ( ItemUse[k] = 0 ) or ( j >= NumItemsperlevel );
      if ItemUse[k] = 0 then (if the item has not been used...)
          if ran0(irndm) < Item[k].Exposure
            then (if the item can be used...)
              begin
                ItemUse[k] := 2;  (mark item as used)
                kTemp := k;
              end
            else (if item cannot be used, based on exposure)
              begin
                ItemUse[k] := 1;  (mark item as requested)
                kTemp := 0;
              end
      else  (if no items are unused or unrequested)
        begin
          kTemp := kDeny; (use last requested (but not used) item in table)
          ItemUse[kDeny] := 2;
        end;
    UNTIL kTemp <> 0;
    NextItem := kTemp;
  end; (PickItem)

Procedure SimOneflcat(var truetheta, esttheta, estvar : real) ;

var
   ttheta, etheta, evar, sstep : real ;
   ii,jj,ir,irprev             : integer ;
   responsediff                : boolean ;
begin
   ttheta := truetheta ;
   etheta := esttheta  ;
   evar := estvar ;
   sstep := step ;
   responsediff := False ;
   for ii:= 1 to itempoolsize do itemuse[ii] := 0 ;
   for ii:= 1 to testlength do begin
       td[ii].itemindex := 0 ;
       td[ii].response := 0 ;
   end ;
   ii := 0 ;
```

BEST COPY AVAILABLE

```
    repeat    {fixed length cat}
       inc(ii) ;
       pickitem(etheta,jj) ;
       td[ii].itemindex := jj ;
       ir := itemresponse(ttheta,item[jj].a,item[jj].b,item[jj].c) ;
       td[ii].response := ir ;
     {control item selection by estimated theta}
       if (responsediff = false) then
          begin
             if ir=1 then etheta := etheta+sstep
                else etheta := etheta-sstep ;
             sstep := sstep*fstep ;
             if ((ii>1)and(ir<>irprev))then begin
               responsediff := true ;
               bayesmodalscore(td,ii,etheta,pav,bmepsd,
                  mlescore,mlevar,fcn,mletroub) ;
               etheta := mlescore ;
               end ;
             irprev := ir ;
             end
       else
          OwenBayesScore(ir,item[jj].a, item[jj].b,item[jj].c,etheta,evar) ;
    until ii = testlength ; {end of fixedlength cat}
    esttheta := etheta ;
    estvar := evar ;
end ; {simoneflcat}

procedure scoreoneflcat(var truetheta:real) ;
    var
       tempr, tempw,temps    : real ;
       it, itx, j            : integer ;
    begin
       numright:= 0.0 ; numwrong := 0.0 ;
       wtright := 0.0 ;  wtwrong := 0.0 ;
       for it := 1 to testlength do begin {accumulate data for scores}
          itx := td[it].itemindex ;
          if td[itx].response = 0 then
             begin
                numwrong := numwrong + 1.0 ;
                wtwrong := wtwrong + weight[itx] ;
             end
             else begin
                numright := numright + 1.0 ;
                wtright := wtright +  weight[itx] ;
             end ;
       end ; {process testlist }
       if numright = 0.0 then
           for j := 2 to 5 do score[j] := -3.0
         else if numwrong = 0.0 then
           for j := 2 to 5 do score[j] := 3.0
         else  begin
           tempr:= wtright/numright ;
           tempw:= wtwrong/numwrong ;
           score[2] := tempr ;
           score[3] := tempw ;
           score[4] := (tempr+partcredit*tempw)/(1.0+partcredit) ;
           score[5] := (wtright+wtwrong)/testlength ;
           end ;
       nitem := testlength ;
       eqnc(td,nitem,esttheta,temps) ;
       score[6] := temps;
```

BEST COPY AVAILABLE

```
      bayesmodalscore(td,nitem,esttheta,pav,mlepsd,
          mlescore,mlevar,fcn,mletroub) ;
      score[1] := mlescore ;
(*      write(outfile,truetheta:8:3);
      for j := 1 to 6 do write(outfile,score[j]:6:2 ) ;
      writeln(outfile) ;
      for it := 1 to testlength do begin
         iteminx := td[it].itemindex ;
         writeln(outfile,it:3,td[it].response:2,
            item[iteminx].b:8:2,weight[iteminx]:6:2) ;
         end ;                                                    *)
  end ; (scoreoneflcat)

  procedure accumonecat ;
     var
       is : integer ;
  begin
  (* process scores for one case *)
     for is := 1 to  numscore do begin  (for each score type)
          meanscore[is] := meanscore[is] + score[is] ;
          varscore[is] := varscore[is] +sqr(score[is]) ;
     end ; (each score type)
  end ; (accumonecat )

procedure Processbatch(var ktheta:integer) ;
  var
    is : integer ;
  begin
    for is := 1 to  numscore do begin
      temp := meanscore[is] ;
      meanscore[is]:= meanscore[is]/numpertheta ;
      batmean[is,ktheta] := meanscore[is] ;
      varscore[is] :=(varscore[is]-meanscore[is]*temp)/numpertheta ;
      temp := sqr(truetheta - meanscore[is])+varscore[is]  ;
      if temp > 0.0 then rmsqbias[is] := sqrt(temp)
         else rmsqbias[is] := 0.0 ;
      if varscore[is] > 0 then varscore[is] := sqrt(varscore[is])
         else varscore[is] := 0.0 ;
      batsd[is,ktheta] := varscore[is] ;
      end ; (is types of score)
  end ; (processbatch)

procedure simbatch(var jtheta:integer) ;

 var
   iss,  it   : integer ;  (ip defined globally)
   temp       : real ;
 begin
   for iss := 1 to  numscore do begin (preset for accum )
      meanscore[iss] := 0.0 ;
      varscore [iss] := 0.0 ;
      rmsqbias [iss] := 0.0 ;
      firsttheta     := 0.0 ;
      end ;  (preset)
   if firstthetacode = 2 then  firsttheta := truetheta
      else firsttheta := starttheta ;
   ip := 0 ;
   repeat
      inc(ip) ;
      esttheta := firsttheta ;
      estvar   := startvar ;
```

BEST COPY AVAILABLE

```
      simoneflcat(truetheta, esttheta, estvar) ;
      scoreoneflcat(truetheta) ;
      accumonecat ;
   until ip = numpertheta ;
   processbatch(jtheta) ;
 end ; {simbatch}

procedure writeheading ;
   var
      j : integer ;
   begin
    writeln(outfile,' output from program ',programlabel:15) ;
    writeln(outfile,'filename: ',outfilename:15,
    '  random start was ',iranstart:7) ;
    writeln(outfile,testlength:2,'-item cats for each of ',
    numpertheta:5,' simulees'  ) ;
    writeln(outfile,'using ',nitems:5,' from item pool ',itemfilename:15) ;
    writeln(outfile,' using exposure control',excode:6:3,
      ' firstthetacode= ',firstthetacode:2);
    writeln(outfile,'amin',amin:6:3,'  bmin',bmin:6:3,'  cmin',cmin:6:3);
    writeln(outfile,'amax',amax:6:3,'  bmax',bmax:6:3,'  cmax',cmax:6:3);
    writeln(outfile, 'linear eqating of weignts  tmi0= ',tmi0:6:3,
           '  tmil=',tmil:6:3) ;
    writeln(outfile,'if tmi < ',lotmi:6:3,' then weight= ',lopoly[0]:6:2,' + ',
       lopoly[1]:6:2,'*tmi + ',lopoly[2]:6:2,'*tmi*tmi') ;
    writeln(outfile,'if tmi > ',hitmi:6:3,' then weight= ',hipoly[0]:6:2,' + ',
       hipoly[1]:6:2,'*tmi + ',hipoly[2]:6:2,'*tmi*tmi') ;
    writeln(outfile,' if ',lotmi:6:3,' < tmi < ',hitmi:6:3,' weight = tmi') ;
    writeln(outfile) ;
    writeln(outfile,'partial credit fraction: ',partcredit:6:3) ;
    writedatetime(outfile) ;
    writeln(outfile) ;
   end ;

procedure makeeqweights ;
   var
      it : integer ;
      tempt, tempp, sumeqw : real ;
   begin
      tempt := lotheta - delta ;
      sumeqw := 0.0 ;
      for it := 1 to numtheta do begin
         tempt := tempt + delta ;
         NormalDensityandProb(tempt,tempp,eqweight[it]) ;
         sumeqw := sumeqw + eqweight[it] ;
         end ;
      for it := 1 to numtheta do begin
         eqweight[it] := eqweight[it] / sumeqw ;
         end ;
   end ; {of makeeqweights}

procedure equatescores ;
  var i,j  : integer ;{i for scores, j for thetas,batmean[i,j] }
  temp , templ: real ;
  begin
    writeln(outfile) ;
    writeln(outfile,' #   score     ',' slope  ',' addconst ') ;
    for i:= 1 to numscore do begin
       meanscore[i] := 0.0 ;
       varscore[i] := 0.0  ;
       for j := 1 to numtheta do begin
```

BEST COPY AVAILABLE

```
            temp := sqr(batmean[i,j]) ;
            meanscore[i] := meanscore[i] + eqweight[j]*batmean[i,j] ;
            varscore[i] := varscore[i] + eqweight[j]*(sqr(batmean[i,j])) ;
            end ;{j}
         varscore[i] := sqrt(varscore[i] - sqr(meanscore[i])) ;
         writeln(outfile,i:3,'  ',
          scorelabel[i]:9,varscore[i]:10:3,meanscore[i]:10:3) ;
         end ; {i}
      writeln(outfile) ;
      writeln(outfile,' #   score     ',' slope   ','addconst  ') ;
      slope[1] := 1.0 ;
      addconst[1] := 0.0 ;
      for i := 2 to numscore do begin
         slope[i] := varscore[1]/varscore[i] ;
         addconst[i] := meanscore[1] - slope[i]*meanscore[i] ;
         writeln(outfile,i:3,'  ',
            scorelabel[i]:9,slope[i]:10:3,addconst[i]:10:3) ;
         for j:= 1 to numtheta do begin
            batsd[i,j] := batsd[i,j]*slope[i] ;
            batmean[i,j] := batmean[i,j]*slope[i] + addconst[i] ;
            end; {J}
         end; {i}
   end ; {equatescores}

procedure writeresults ;
   var
     i,j : integer ;
   begin
     writeln(outfile,' score means');
     write(outfile,'true theta ');
     for i := 1 to numscore do write(outfile,scorelabel[i]:9,' ') ;
     writeln(outfile) ;
     truetheta := lotheta - delta ;
     for j := 1 to numtheta do begin
        truetheta := truetheta + delta ;
        write(outfile,truetheta:8:3,'   ') ;
        for i := 1 to numscore do write(outfile,batmean[i,j]:9:3) ;
        writeln(outfile) ;
        end ; {j}
     writeln(outfile,'  standard deviations');
     write(outfile,'true theta ');
     for i := 1 to numscore do write(outfile,scorelabel[i]:9,' ') ;
     writeln(outfile) ;
     truetheta := lotheta - delta ;
     for j := 1 to numtheta do begin
        truetheta := truetheta + delta ;
        write(outfile,truetheta:8:3,'   ') ;
        for i := 1 to numscore do write(outfile,batsd[i,j]:9:3) ;
        writeln(outfile) ;
        end ; {j}
     writeln(outfile,'  root mean square errors');
     write(outfile,'true theta ');
     for i := 1 to numscore do write(outfile,scorelabel[i]:9,' ') ;
     writeln(outfile) ;
     truetheta := lotheta - delta ;
     for j := 1 to numtheta do begin {RMS}
        truetheta := truetheta + delta ;
        write(outfile,truetheta:8:3,'   ') ;
        for i := 1 to numscore do begin  {rmsj}
           temp:= sqr(truetheta - batmean[i,j])+ sqr(batsd[i,j]) ;
           temp := sqrt(temp) ;
```

BEST COPY AVAILABLE

```
            write(outfile,temp:9:3) ;
            end ; {rmsj}
         writeln(outfile) ;
         end ; {RMS}
   end; {writeequateresults }

{ Main }
Begin
   createoutfile(outfile) ;
   irndm := randomize0 ; {needed for itemresponse in irtlibw }
      {gets a five digit integer from user for random seed}
   getitemdata ;
   nitems := itempoolsize ;
   writeln(nitems:4,' obtained from file ',itemfilename:30) ;
   makeinfotable ;
   writeln('infotable made') ;
   makeitemweights ;
   makeeqweights ;
 writesomeitems;
   hitheta := lotheta +(numtheta-1)*delta ;
   truetheta := lotheta - delta ;
   writeheading ;
   writeln('simulating ',testlength:2,'-item cats for each of ',
      numpertheta:5,' simulees') ;
   for itheta := 1 to numtheta do begin
       truetheta := truetheta + delta ;
       writeln('working on theta = ',truetheta:7:3) ;
       simbatch(itheta) ;
   end ;  {end all batches}
   writeln(outfile,'raw means, sd, rms');
   writeresults ;
   equatescores ;
   writeln(outfile,'equated means, sd, rms') ;
   writeresults ;
   writedatetime(outfile) ;
   close (outfile) ;
   writeln('finished writing file ',outfilename:12,' with',
      numtheta:5,' thetas.' ) ;
 end.  {of simscr24 2/15/97 }
```

BEST COPY AVAILABLE

C2: IRT subroutines

```
{ CAT\irtlibw.pas  modified 6/25/95 and 1/12/96
 Library of functions and procedures that will be used in CAT simulations  }
unit irtLibw;

interface

uses  wincrt;

var
   irndm, iranstart, itemp : integer ;
   Ran0Y : real ;
   Ran0V : array [1..97]OF real ;

function randomize0 : integer ;

function ran0(var idum: integer): real ;
      {from Numerical Recipes; equivalent to random, but better }

function RandomInt (var m : integer ): integer ;

function RandomNormal (var mean,sd : real): real ;

procedure NormalDensityAndProb ( var x,p,density : real);
      { normal density and prob. for given norm deviate }

function ZofP (p : real) : real;
      { normal deviate of given prob }

function BoundedExp (x : real) : real;
      { bounded exponential function }

function ProbIRT (Theta,a,b,c : real) : real;
      { gives irt prob for Theta, a, b, c }

function ItemResponse (Theta,a,b,c : real) : integer ;
      { irt response to item k for Theta; result 0 or 1 }

function Info (Theta,a,b,c : real) : real ;
      { Info for item k at Theta }

function ThetaForMaxInfo (a,b,c : real) : real ;
      { Theta at which item k has max info }

function thetaforp50 (a,b,c : real) : real ;
      { theta for which prob(resp = 1) = 0.50 }
{=============================================================================}
                              implementation
{=============================================================================}
    function randomize0 : integer;
      begin
         write('type a 4-digit integer  ') ;
         readln (iranstart) ;
        itemp := abs(iranstart) ;
         while itemp > 10000 do
            itemp := itemp - 10000 ;
         randomize0 := - itemp;
      end ;
```

BEST COPY AVAILABLE

```
(*usage:program that calls ran0(idum) must have declared
   var idum or equiv.
     and must issue idum := randomize0  at start to set up randseed*)

    function ran0(var idum: integer): real ;
      {from Numerical Recipes }
      var
         dum: real ;
         j: integer ;

      begin
         if idum < 0 then begin
            randseed := -idum ;
            idum := 1 ;
            for j := 1 to 102 do
                dum := random ;
            for j := 1 to 97 do
                ran0V[j] := Random ;
            Ran0Y := Random ;
         end ;
         j := 1 + trunc(97.0*ran0Y) ;
         if(j > 97) OR (j < 1) then begin
              writeln('pause in routine ran0') ;
              readln ;
         end ;
         ran0Y := ran0V[j] ;
         ran0 := ran0Y ;
         ran0V[j] := random ;
      end ; {ran0}

{-------------------------------------------------------------------------}
 function RandomInt (var m : integer ): integer ;
   var x: real ;
   begin
     x := m*ran0(irndm) ;
     RandomInt := trunc(x) + 1
   end ;
{-------------------------------------------------------------------------}
function RandomNormal (var mean, sd : real ) : real ;
   var x : real ;
   begin
     x := ran0(irndm) ;
     x := zofp(x) ;
     RandomNormal := sd*x + mean ;
   end ;


   procedure NormalDensityAndProb (var x,p,density : real);
      { x : normal deviate (input)
        p : output probability of a value less than x
        density : normal density at x }
     var temp, xx : real;
     begin
       xx := -x*x/2;
       density := 0.3989423*exp(xx);
       temp := 1/(1+0.2316419*abs(x));
       p := 1-density*temp*(((1.330274*temp-1.821256)*temp+1.781478*
            temp-0.3565638)*temp+0.3193815);
       if x < 0
         then p := 1-p;
```


BEST COPY AVAILABLE

```
    end; {NormalDensityAndProb}

{ ------------------------------------------------------------------------}
  function ZofP (p : real) : real ; {computes normal deviate of given prob}
    var pp, tp, tn, td, temp : real;
        ip : integer;
    begin
      if p < 0 then temp := -5.0
        else
          if p > 1 then temp := 5.0
            else
              begin {0<=p<=1}
                if p < 0.5
                  then
                    begin {0<=p<0.5}
                      pp := p;
                      ip := -1;
                    end {0<=p<0.5}
                  else
                    begin {0.5<=p<=1}
                      pp := 1-p;
                      ip := 1;
                    end; {0.5<=p<=1}
                tp := sqrt(-2*ln(pp));
                tn := (0.0103280*tp+0.802853)*tp+2.515517;
                td := ((0.001308*tp+0.189269)*tp+1.432788)*tp+1;
                temp := tp-tn/td;
                if ip = -1 then temp := -temp;
              end; {0<=p<=1}
      ZofP := temp;
    end; {ZofP}

{ ------------------------------------------------------------------------}
  function BoundedExp (x : real) : real ;
    const
      Max =  29;
      Min = -87;
    var
      y  : real;
    begin
      if x < Min then y := Min
        else
          if x > Max then y := Max
            else y := x;
      BoundedExp := exp(y);
    end; {BoundedExp}

{ ------------------------------------------------------------------------}
  function ProbIRT  (Theta,a,b,c : real) : real ;
    var y, v, p : real;
    begin
      y := 1.702*a*(Theta-b);
      v := BoundedExp(y);
      p := (c+v)/(1+v);
      if p > 1.0 then p := 1.0 ;
      if p < 0.0 then p := 0.0 ;
      ProbIRT := p ;
end; {ProbIRT}

{ ------------------------------------------------------------------------}
  function ItemResponse (Theta,a,b,c : real) : integer ;
```

```
    {ItemResponse is 0 for fail, 1 for pass on item k at Theta}
    var pp : real;
    begin
      pp := ProbIRT(Theta,a,b,c);
      ItemResponse := trunc(pp + ran0(irndm)) {random} ;
    end; {ItemResponse}

{ ---------------------------------------------------------------------------}
  function Info (Theta,a,b,c : real)  : real ;
    var da, y, z, zc, temp : real;
    begin
      if (a <= 0.0) or (c < 0.0) then Info := 0.0
         else begin
      da := 1.702*a;
      y := da*(b-Theta);
      z := BoundedExp(y);
      zc := z*c;
      temp := da/(1.0 + z);
      Info := temp*temp*(z-zc)/(1.0 + zc);
      end ;
    end; {Info}

{ ---------------------------------------------------------------------------}
  function ThetaForMaxInfo (a,b,c : real) : real ;
    var tmi : real ;
    begin
      if (a > 0.0) and (c >= 0.0) then
         tmi:= B+(ln(0.5+sqrt(0.25+2*C)))/(1.7*A)
       else tmi := 99.0 ;
      ThetaForMaxInfo := tmi ;
    end; {ThetaForMaxInfo}

{ ---------------------------------------------------------------------------}
  function thetaforp50 (a,b,c : real) : real ;
    { theta for which prob(resp = 1) = 0.50 }
    var p50 : real ;
    begin
      if (a<= 0.0) or (c >= 0.5) then thetaforp50 := 0.0 else
        thetaforp50 := b + (ln(1.0 - 2.0*c))/(1.7*a) ;
    end ;
{ ---------------------------------------------------------------------------}
end.
```

BEST COPY AVAILABLE

C3: Scoring programs

```
unit scorew ;  {revised 6/27/95}

interface

uses wincrt, irtlibw, testdefw ;

const
   bug : integer = 0 ;

var
   scoreout : text ;

procedure OwenBayesScore(var iresp : integer; var a,b,c,curt,curv : real ) ;
  {Note: also in program simflcat }
function itemlocation(var itemindex,poolsize : integer):integer ;

procedure eqnc(var tl:testlist;var ntest:integer;
               var intheta, outtheta : real) ;

procedure BayesModalScore (var tl: testlist;
    var nlist : integer;
    var intheta, pav, psd, outtheta, thetavar, funct: real;
    var troubleout : integer ) ;

{=============================================================================}
implementation
{=============================================================================}
 const
    mintheta : real = -3.0 ;
    maxtheta : real =  3.0 ;
    dtheta1  : real =  0.5 ;
    dtheta2  : real =  0.1 ;
    limit2   : real =  0.5 ;

 var
    (* item : itempool  (in irtlib) *)
    bnpool,bnlist,trouble,nscan          : integer ;
    bintheta,bpav,bpsd                    : real ;
    pbtheta,pbfunct,pbdf,pbddf            : real ;
    lotheta,hitheta,dtheta : real ;
    bestt,bestf,bestdf,bestddf    : real ;
    oldt,oldf,olddf,oldddf        : real ;

 { Owen's-bayes sequential scoring of CAT by updating theta and its variance}
 procedure OwenBayesScore (var iresp : integer; var a,b,c,curt,curv : real ) ;

    { iresp is 1,0 response to item with params a,b,c
      at call, curt and curv are current theta and its variance
      at end curt and curv have been updated a la Owen (NPRDC version)  }
     var
       ss,s,xx,pp,dd,q,r : real ;
     begin  {OwenBayesScore}
          ss := curv + 1.0/(sqr(a)) ;
          s := sqrt (ss) ;
          xx := (curt - b)/s ;
          NormalDensityAndProb(xx,pp,dd) ;
          q := 1. - pp ;
```

BEST COPY AVAILABLE

```
            if iresp = 1 then
               r := dd/((1./(1.-c))-q)
             else
               r := -dd/q ;
            curt := curt+r*curv/s ;
            curv := curv-sqr(curv)*r*(r+xx)/ss ;
            if curv < 0.0001 then curv := 0.0001 ;
         end ; {of OwenBayesScore}

function itemlocation(var itemindex,poolsize : integer):integer ;
   var
      ii,jj : integer ;
   label 77 ;
   begin
      ii:= 0 ;
      if item[itemindex].id = itemindex then ii := itemindex else
         for jj := 1 to poolsize do
            if item[jj].id = itemindex then begin
               ii:= jj ;
               goto 77;
            end;
   77:   itemlocation := ii ;
   end ;

procedure fdf (var tl:testlist;var ntest:integer; var theta,sp,sdp:real) ;
   var
     i,j,jj       : integer ;
     tp, tdp, aa, bb, cc, pp      : real  ;
begin
   tp := 0.0 ;
   tdp := 0.0 ;
   if ntest > 1 then
   for i := 1 to ntest do begin
      jj:= tl[i].itemindex ;
        j:= itemlocation(jj,itempoolsize) ;
      aa := item[j].a ;
      bb := item[j].b ;
      cc := item[j].c ;
      pp := probirt(theta, aa,bb,cc) ;
      if tl[i].response = 1 then tp := tp + 1.0 ;
      tp := tp -  pp ;
      tdp := tdp - 1.702*aa*(1.0 - pp)*(pp - cc)/(1.0 - cc) ;
      end ;
    sp := tp ;
    sdp := tdp ;
 end ;

 procedure eqnc(var tl:testlist;var ntest:integer;
                var intheta, outtheta : real) ;
    var
       i,j : integer ;
       theta, newtheta, dtheta, fcn, dfcn, diff : real  ;
       flag : boolean ;
    const
       eps = 0.0001 ;
    begin
       flag := TRUE ;
       diff := 2.0*eps ;
       newtheta := intheta ;
       if ntest < 1 then flag := FALSE ;
       while (abs(diff) > eps) and (flag = TRUE) do begin  {Newton's method }
```

BEST COPY AVAILABLE

```
            theta := newtheta ;
            fdf(tl,ntest,theta,fcn,dfcn) ;
            if dfcn <>0.0 then newtheta := theta - fcn/dfcn
              else begin
                newtheta:= theta;
                flag := FALSE;
                end ;
            diff := newtheta - theta ;
            if abs(newtheta) > 3.0 then flag := FALSE ;
            end ;
          if flag = FALSE then
            if newtheta > 3.0 then outtheta := 3.0
              else if newtheta < -3.0 then outtheta := -3.0
              else
              outtheta := theta
           else
            outtheta := newtheta ;
      end ; {eqnc}

procedure postd(var tl : testlist) ;  {used by Bayesmodalscore }
  {uses item, testlist as in testdefw }
  {needs to be set: pbtheta,bpav,bpsd,trouble;
              sets: pbfunct,pbdf,pbddf,trouble(if necessary) }

  {computes, at a specified theta, either likelihood function(if psd=0.) or
     posterior density function with normal prior mean pav, s.d. psd,
     for items and responses on tl, of length nlist,
     which used items in ip, of length npool.
     Output is function, likelihood or posterior density at theta (not log
density)
     and df, ddf, 1st and 2nd derivatives of log likelihood or density
     tilt is false if ok, true if trouble}

  var
     i,j,it : integer ;
     prod,dlp,ddlp,aa,cm,u,v,av,auv,p,q,dp,dpp,z,d : real ;

  label
     98,90 ;

  begin  {postd}
     prod := 1. ;
     dlp  := 0. ;
     ddlp := 0. ;
     if bnlist <= 0 then goto 98 else begin {test processing}
        for it := 1 to bnlist do begin {testloop}
           j := tl[it].itemindex ;
           i := itemlocation(j,bnpool) ;
           if (i<=0) OR (i>bnpool) then goto 98 else begin {calc}
              aa := 1.7*item[i].A ;
              cm := 1. - item[i].C ;
              u := 1./(1.+ boundedexp(aa*(pbtheta - item[i].B))) ;
              q := cm*u ;
              v := 1. - u ;
              av := aa*v ;
              auv := av*u ;
              if tl[it].response = 1 then
                 begin {right}
                    p:=1.-q ;
                    dp:=cm*auv ;
                    dpp :=dp/p ;
```

```
                    prod := prod*p ;
                    dlp := dlp + dpp ;
                    ddlp := ddlp + dpp*(aa*(u-v)-dpp) ;
                 end  {right}
              else begin  {wrong}
                 prod := prod*q ;
                 dlp  := dlp-av ;
                 ddlp :=ddlp-aa*auv ;
              end; {of right or wrong}
           end;{ of calc}
        end;{of testloop}
     end; {of item processing}
     {now add contribution of prior, if psd nonzero}
     if bpsd>0 then begin {exit set }
        z := (pbtheta-bpav)/bpsd ;
        NormalDensityandProb(z,p,d) ;
        pbfunct := prod*d ;
        pbdf := dlp - z/bpsd ;
        pbddf :=ddlp - 1./sqr(bpsd) ;
     end {of Bayesmodal exit}
     else begin
        pbfunct := prod ;
        pbdf := dlp ;
        pbddf := ddlp ;
     end ; {of Maxlikelihood exit and exit set}
     goto 90 ;

98: trouble := 4 ; {output same as input if trouble with testlist or items}

90: if bug > 0 then writeln(scoreout,'pd:  ',bnlist:3,pbtheta:8:5,
        ' f=',pbfunct:12:6,' df=',pbdf:12:6,' ddf=',pbddf:12:6,
        ' trouble= ',trouble:2) ;
end; { of postd}

procedure scantheta (var tl : testlist) ;
    {uses item as itempool as in irtlib}
    {needs to be set: nscan,pbtheta,trouble; increments scan,
    sets to best found: pbtheta,pbfunct,pbdf,pbddf,trouble(if necessary)}

label 88, 10 ;

begin
10: if nscan = 1 then begin {set first}
      lotheta := mintheta ;
      hitheta := maxtheta ;
      dtheta  := dtheta1  ;
   end {set first}
   else begin {set second}
      lotheta := pbtheta - limit2 ;
      hitheta := pbtheta + limit2 ;
      dtheta  := dtheta2 ;
   end ;{set second}
   bestf := 0.0 ;
   pbtheta := lotheta ;
   repeat
      postd(tl) ;
      if trouble <> 0 then goto 88 ;
      if pbfunct >= bestf then begin {savebest}
         bestt := pbtheta ;
         bestf := pbfunct ;
         bestdf := pbdf ;
```

BEST COPY AVAILABLE

```
            bestddf := pbddf ;
         end ;{savebest}
         pbtheta := pbtheta + dtheta ;
      until (pbtheta > hitheta) ;
      pbtheta := bestt ;
      pbfunct := bestf ;
      pbdf := bestdf ;
      pbddf := bestddf ;
      inc(nscan) ;
      if (bestddf > 0.0) and (nscan = 2) then goto 10 ; {try again}

88: if nscan >= 3 then trouble := 3 ;
      if trouble <> 0 then
         if pbtheta <= mintheta then begin
            pbtheta := mintheta ;
            trouble := 2 ;
         end
         else if pbtheta >= maxtheta then begin
            pbtheta := maxtheta ;
            trouble := 2 ;
      end ;
end ; {of scantheta}

procedure BayesModalScore (var tl: testlist; {from testdefw}
      var nlist : integer; {number of pairs on testlist}
         {assumes item (an itempool), and itempoolsize from testdefw}
      var intheta, pav, psd, outtheta, thetavar, funct: real;
       {intheta is estimated theta at input }
       {pav,psd are mean and sd of prior theta dsn - psd=0 means MLE called for}
       {at output, outtheta,thetavar are estimated t,var(t)}
        {function is max found by postd}
      var troubleout : integer ) ;
{ At exit, trouble has value indicating quality of score:
     0 = still trying (should not be an output value )
     1 = good estimate (delta too small, so zeroed in )
     2 = theta out of bounds
     3 = max.no. of iterations exceeded
     4 = no test input )
{after Divgi's fortran program ,1986}

const
   maxiter   : integer = 10 ;
   criterion : real = 0.0001 ;
   maxthvar  : real = 99.0  ;
   minvalue  : real = 1.E-18;
   stepsize  : real = 0.2   ;
   one       : real = 1.0   ;

label 90, 91, 92, 93, 94, 98, 99 ;

var
   iter, try : integer ;
   delta : real ;

begin {BayesModalScore}
   pbtheta := intheta ; {presets}
   oldt := pbtheta ;
   oldf := 0.0 ;
   olddf := 0.0 ;
   oldddf := 0.0  ;
   trouble := 0 ;
```

BEST COPY AVAILABLE

```
   bnpool := itempoolsize ;
   bnlist := nlist ;
   bpav := pav ;
   bpsd := psd ;
   nscan := 1 ;

   if bnlist <= 0 then
      begin
      trouble := 4 ;
      goto 94 ; {if test empty}
   end ;

   postd(tl) ;
   if pbddf > 0.0 then begin {way off base: search}
   { neg. 2nd deriv. means min, not max; seek df sign change }
      scantheta(tl) ;
      if trouble <> 0 then goto 90 ;
   end ; {way off base }

{now ddf < 0 ; iterate (first postd call made); get delta }
   for iter := 1 to maxiter do begin {Newton Iteration}
      if iter > 1 then postd(tl) ;
      if pbddf > 0.0 then begin {pos. ddf found after earlier OK}
         pbtheta := oldt ; {must be an oldt if got here}
         scantheta(tl) ;
         if trouble <> 0 then goto 90 ;
      end; {pos ddf found}

{set step}
      if(iter>1) and (oldf>pbfunct) then  {went too far}
         if abs(pbdf - olddf) < minvalue then {avoid underflow }
            olddf := 0.0
         else
            olddf := (pbdf - olddf)/(pbtheta - oldt)
         {end of went too far}
         {in this case, newdf and newddf not installed as old}
      else begin {ordinary step; save}
         oldt   := pbtheta ;
         oldf   := pbfunct ;
         olddf  := pbdf ;
         olddf := pbddf ;
      end; {of ordinary step; save}
      {end of set step }

      if abs(olddf) > abs(olddf) then {limit step size}
         if olddf < 0 then delta := -1.0 else delta := +1.0
      else
         delta := -olddf/olddf ;  {standard Newton method}
      {end of limit step size}

      pbtheta := oldt + delta ;  {actually make step }

      if bug > 0 then writeln(scoreout,'BM: ',iter:2,oldt:8:5,oldf:12:6,
         olddf:12:6,olddf:12:6,delta:6:3,pbtheta:8:5) ;

      if(abs(pbtheta) > abs(maxtheta)) then begin {out of bounds}
         trouble := 2 ;
         if pbtheta <= mintheta then pbtheta := mintheta ;
         if pbtheta >= maxtheta then pbtheta := maxtheta ;
         goto 92 ;
      end ;{of out of bounds}
```

BEST COPY AVAILABLE

```
        if abs(delta) < criterion then begin {step too small}
           trouble := 1 ; {no trouble at all!}
           goto 98 ;
        end ;{of step too small}

     end ; {of Newton iteration}
     {continue until (iter > maxiter) i.e. too many iterations
        or (abs(newt) > abs(maxtheta)) i.e. step goes out of bounds
        or (delta < criterion) i.e. step size is too small to bother with}
     trouble := 3 ;{max iterations exceeded }
     goto 98 ;

  90: if trouble <= 1 then goto 98 ;
     if trouble = 2 then goto 92 ; {out of bounds}
     if trouble = 3 then goto 93 ; { poor estimate }
     if trouble >= 4 then goto 94 ;{bad input }

  92: outtheta := pbtheta ;
     funct := pbfunct ;
     oldddf := 1.0 ;
     goto 99 ;
  93: outtheta := pbtheta ;
     funct := pbfunct ;
     oldddf := pbddf ;
     goto 99 ;
  94: goto 98 ;
  98: outtheta := oldt ; {normal exit}
     funct := oldf ;
  99: if oldddf >= -criterion then {bad thetavar}
        if psd = 0.0 then
           thetavar := maxthvar   {bad ml exit}
        else
           thetavar := sqr(psd)   {bad bayes exit}
     else
        thetavar := -1./oldddf  ; {good thetavar exit}
     troubleout := trouble ;
 end ; { of BayesModalScore }
end.  {of scorew unit}
```

C4: General programs and test definitions

```
unit testdefw; {6/27/95}

interface

uses
   wincrt,
   windos,
   irtlibw ;

const
  {for info table dimesnions}
   Numlevels          =  57 ; {max num of theta levels}
   Numitemsperlevel   =  50 ; {max num of items per theta level}
   maxtheta            = 3.5  ;
   mintheta            =-3.5  ;
  {for item pool}
     Maxpoolsize       =  300 ;
   {for cat data}
   maxtestlength       =  50 ;
type
    ItemRecord = record
       ID       : integer;  {Arbitrary item ID number.      }
       A, B, C  : real;     {Item parameters in 3-PL model.}
       Category : integer;  {Type of item.                  }
       Exposure : real;     {0.<Prob<=1. that item MAY be exposed}
    end; {of itemrecord definition}
    ItemPool = array [1 .. maxpoolsize] of ItemRecord;

    testitem = RECORD
       itemindex : integer ;
       response  : integer ;
    end; {of testitem record def.}
    testlist = ARRAY[1..maxtestlength] OF testitem ;

var ItemFile, TestFile, Outfile                        : text   ;
    ItemFileName, TestFileName, OutFileName            : string ;
    itempoolsize  : integer ;
    Item : itempool ;
    itemuse : array[1..maxpoolsize] of integer ;
    infotable : array[1..numlevels,1..numitemsperlevel] of integer;
    grain, invgrain , c1, excode : real ;
     nitems : integer ;
 {1=no exposure control; 2:ex=.5-1.0 }
     amin, amax,bmin, bmax,cmin, cmax :real         ;

procedure GetString (Message : string;
                     var Info : string);
procedure GetInteger (Message : string;
                      var Info : integer);
procedure OpenItemFile;
procedure CreateOutFile (var OutFile : text);
procedure GetItemData;
(*procedure WriteItemData(var itemfileout : text; var nitems : integer ) ;*)
(* procedure Beep; *)
procedure WriteDateTime (var Outfile : text);
procedure OpenTestFile;
procedure MakeInfoTable ;{ make info table with all items }
```

BEST COPY AVAILABLE

```
procedure SaveInfoTable (var OutFile : text; OutFileName : String);
  {saves info table to outfile}
function ThetaOfLevel (jj : integer ): real ; {grain control: Theta of jjth
level}
function LevelOfTheta (Theta : real) : integer; {grain control:Level of Theta}

{=============================================================================}
                              implementation
{=============================================================================}
var Count, m, i, j : integer;

procedure GetString (Message : string;
                     var Info : string);
Begin
  write ( Message );
  readln ( Info );
end; {GetString}

{-----------------------------------------------------------------------------}
procedure GetInteger (Message : string;
                      var Info : integer);
begin
  write ( Message );
  readln ( Info );
end; {GetInteger}

{-----------------------------------------------------------------------------}
procedure OpenItemFile;                    { Use of IOCode in this way      }
var IOCode : integer;                     { prevents system from crashing  }
begin                                     { if InFile can not be opened    }
  (*$I-*)                                 { successfully                   }
  repeat
    GetString ('Item File Name: ', ItemFileName);
    assign (Itemfile, ItemFileName);
    reset ( ItemFile);
    IOCode := IOResult;
    if IOCode <> 0 then
      writeln ('File ', ItemFileName, ' does not exist.  Try again.');
  until IOCode = 0;
  (*$I+*)
end; {OpenInFile}

{-----------------------------------------------------------------------------}
procedure OpenTestFile;                    { Use of IOCode in this way      }
var IOCode : integer;                     { prevents system from crashing  }
begin                                     { if TestFile can not be opened  }
  (*$I-*)                                 { successfully                   }
  repeat
    GetString ('Test File Name: ', TestFileName);
    assign (TestFile, TestFileName);
    reset ( TestFile);
    IOCode := IOResult;
    if IOCode <> 0 then
      writeln ('File ', TestFileName, ' does not exist.  Try again.');
  until IOCode = 0;
  (*$I+*)
end; {OpenTestFile}
{-----------------------------------------------------------------------------}
procedure CreateOutFile (var OutFile : text);

var IOCode : integer;
```

```
begin
  (*$I-*)
  repeat
    GetString ('Output File Name: ', OutFileName);
    assign (OutFile, OutFileName);
    rewrite ( OutFile);
    IOCode := IOResult;
    if IOCode <> 0 then
      writeln ('File ', OutFileName, ' cannot be created.  Try again.');
  until IOCode = 0;
  (*$I+*)
end; {CreateOutFile}

{-------------------------------------------------------------------------}
procedure ReadItemData; {assumes ItemFIle is open, closes it at end}
var Count  : integer;
begin
  writeln ('Getting item information... ');
  readln(itemfile) ;
  readln(itemfile,amin,amax,bmin,bmax,cmin,cmax,excode) ;
  Count := 0;
  while (not eof (ItemFile)) and (Count <> maxpoolsize) do
    begin
      inc (Count);
      readln (ItemFile, Item[Count].ID, Item[Count].A, Item[Count].B,
                        Item[Count].C, Item[Count].Category,
                        Item[Count].Exposure);
      if Item[Count].ID = 0 then dec(Count) ;
    end; {while not eof and count<>maxpoolsize}
  close (ItemFile);
  itempoolsize := Count;
  writeln(' item pool size = ',itempoolsize:5) ;
end; {GetItemData}
{-------------------------------------------------------------------------}
procedure GetItemData;
begin
   OpenItemFile ;
   ReadItemData ;
end ; {getItemData }

{-------------------------------------------------------------------------}
(*
procedure WriteItemData(var itemfileout : text; var nitems : integer ) ;
var  count : integer ;
begin
  if itempoolsize > maxpoolsize then itempoolsize := maxpoolsize ;
  writeln ('Writing item information... ');
  writeln(itemfile) ;

writeln(itemfile,amin:6:2,amax:6:2,bmin:6:2,bmax:6:2,cmin:6:2,cmax:6:2,excode:3)
;
  for count := 1 to itempoolsize do
     writeln(itemfileout, Item[Count].ID, Item[Count].A, Item[Count].B,
        Item[Count].C, Item[Count].Category, Item[Count].Exposure);
end; {WriteItemData}
*)
{-------------------------------------------------------------------------}
procedure WriteDateTime (var Outfile : text) ;
  var yr,mo,dy,dow,hr,mi,sc,sc100 : word ;
begin
```

```
   GetTime(hr,mi,sc,sc100) ;
   GetDate(yr,mo,dy,dow) ;
   Writeln(Outfile,'date=',mo:2,'/',dy:2,'/',yr:4,
     ' at ',hr:2,':',mi:2,':',sc:2,'.',sc100:2) ;
end; {of WriteDateTime}
{-------------------------------------------------------------------------}
(*
procedure Beep;
begin
  sound (1500);
  delay(150);
  nosound;
end; {Beep}
*)
{-------------------------------------------------------------------------}
procedure MakeInfoTable;
  {makes info table using the itempoolsize items in the item pool}
  var i, j, k, jj, kk, itemp : integer;
      Theta : real;
      InfoFile  : text;
      infolist  : array[1..maxpoolsize] of real ;
      indexlist : array[1..maxpoolsize] of integer ;
  begin
    grain := (maxtheta-mintheta)/(numlevels-1) ;
    invgrain := 1.0 / grain;
    c1 := (mintheta*invgrain) - 1.5 ;
    writeln ('Making info table  ');
    for j := 1 to Numlevels do begin {for each theta level}
       Theta := ThetaOfLevel(j); {determine theta}
       for k := 1 to ItemPoolSize do begin {get info at level for each item}
          IndexList[k] := item[k].id;
          InfoList [k] := Info(Theta,Item[k].a,Item[k].b,Item[k].c);
       end; {get info for itempool}
       for k := 1 to (numitemsperlevel-1) do begin {sort by info hi-lo}
           for kk := (k+1) to itempoolsize do begin
              if infolist[(indexlist[kk])] > infolist[(indexlist[k])] then begin
{switch}
                 itemp := indexlist[kk] ;
                 indexlist[kk] := indexlist[k] ;
                 indexlist[k] := itemp ;
              end ; {switch indexlist only}
           end ; {kk}
       end ; {k. sorting finished}
       for k := 1 to numitemsperlevel do InfoTable[j,k] := IndexList[k];
{install}
    end ; {j. Finished all theta levels}
  end; { MakeInfoTable }

{ ------------------------------------------------------------------------}

procedure SaveInfoTable (var OutFile : text; OutFileName : String);
  {saves info table to outfile}
  var  j, k : integer;
       temp : real ;
  begin
  writeln ('Saving Info Table in ', OutFileName);
      begin
        writeln (OutFile,'Info Table ');
        writeln (OutFile);
        writeln (OutFile, 'Theta   Item Indices');
        for j := 1 to Numlevels do begin
```

```
            temp := thetaoflevel(j) ;
            writeln (OutFile, temp:6:2);
            for k := 1 to 15 do
                write (OutFile, InfoTable[j,k]:4);
            writeln (OutFile); {end jth line in file}
        end ; {for j}
      end ; {table done}
  end; {SaveInfoTable}

{ ------------------------------------------------------------------------}

function ThetaOfLevel (jj : integer ): real ; {grain control: Theta of jth
column}
begin
  if jj < 1  then jj := 1;
  if jj > numlevels then jj := numlevels;
  ThetaOfLevel := (jj-1)*grain+mintheta; {1(1)numlevels =
mintheta(grain)maxtheta}
end; {ThetaOfLevel}

{ ------------------------------------------------------------------------}

function LevelOfTheta (Theta : real) : integer; {grain control: col of Theta}
var jj : integer;
begin
  jj := trunc((Theta*invgrain)- c1 );
  if jj < 1  then jj := 1;
  if jj > numlevels then jj := numlevels ;
  LevelOfTheta := jj;     {mintheta(grain)maxtheta := 1(1)numlevels}
end; {LevelOfTheta}

end.  {unit testdefw }
```

U.S. Department of Education
Office of Educational Research and Improvement (OERI)
National Library of Education (NLE)
Educational Resources Information Center (ERIC)

NOTICE

Reproduction Basis

[X] This document is covered by a signed "Reproduction Release (Blanket)" form (on file within the ERIC system), encompassing all or classes of documents from its source organization and, therefore, does not require a "Specific Document" Release form.

[ ] This document is Federally-funded, or carries its own permission to reproduce, or is otherwise in the public domain and, therefore, may be reproduced by ERIC without a signed Reproduction Release form (either "Specific Document" or "Blanket").